# III FUNKTIONSWEISE DES PROZESSORS

S. BARTELS, 28.5.2018

## 1. Von der Binärzahl zum Transistor

Computer arbeiten mit dem Binärsystem, das auch als Dualsystem bezeichnet wird, und ihre Prozessoren bestehen aus vielen Transistoren. Wir diskutieren im Folgenden, wie arithmetische Operationen im Binärsystem durchgeführt werden können, wie sich diese durch logische Operationen darstellen lassen und wie sich logische Operationen mit elektrischen Schaltungen realisieren lassen.

## 2. Binärdarstellung natürlicher Zahlen

Jede natürliche Zahl inklusive Null lässt sich als Summe von Potenzen der Zahl 2 darstellen, das heißt für jedes $\ell \in \mathbb{N}_0$ existieren eine natürliche Zahl $k \in \mathbb{N}$ und Koeffizienten $b_0, b_1, \dots, b_k \in \{0, 1\}$, so dass

$$\ell = b_k \cdot 2^k + \cdots + b_1 \cdot 2^1 + b_0 \cdot 2^0.$$

Schreibt man die Zahlen $b_0, b_1, \dots, b_k$ hintereinander, so erhält man die *Binärdarstellung* von $\ell$, beispielsweise

$$13 = 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 \equiv 1101$$

Das Dezimalsystem lässt sich übrigens genau so interpretieren, wenn statt der Basis 2 die Basis 10 verwendet wird. Jede Ziffer der Binärdarstellung bezeichnen wir als *Bit*. Fixieren wir die maximale Anzahl $k$ zulässiger Bits, so können wir natürliche Zahlen im Bereich $0, 1, \dots, 2^k - 1$ darstellen. Die Darstellung von Dezimalzahlen mit vorgegebener Anzahl $m$ von Nachkommastellen lässt sich auf die Darstellung natürlicher Zahlen zurückfüren, wenn wir alle solchen Dezimalzahlen mit $10^m$ multiplizieren, zur Berücksichtigung negativer Zahlen können wir ein Bit zur Speicherung des Vorzeichens verwenden. Insgesamt lassen sich so arithmetische Operationen mit Dezimalzahlen auf das Rechnen mit Binärzahlen zurückführen.

## 3. Rechnen im Binärsystem

Die Addition zweier Binärzahlen erfolgt schriftlich mit Übertrag, im Beispiel $13 + 6$ ergibt sich:

$$\begin{array}{rr} & 1101 \\ + & 0110 \\ \hline \text{Ü} & 1100 \\ \hline & 10011 \end{array}$$

Die Subtraktion wird auf die Addition zurückgeführt. Dazu werden Minuend und Subtrahend durch Auffüllen mit Nullen auf gleiche Bitlänge gebracht und das additive Inverse des Subtrahenden wie folgt gebildet. Zunächst werden alle Bits invertiert, das heißt aus $8 \equiv 1000$ wird 0111, und anschließend wird 1 addiert, das heißt aus 0111 wird 1000. Diese Zahl wird zum Minuenden addiert und anschließend das höchste Bit in der Summe gelöscht. Für die Rechnung $14 - 8$ ergibt sich in Binärdarstellung also die Addition:

$$\begin{array}{rr} & 1110 \\ + & 1000 \\ \hline \text{Ü} & 1000 \\ \hline & \cancel{1}0110 \end{array}$$

Zur Multiplikation einer Binärzahl mit einer Potenz der Zahl 2, werden durch Anhängen von Nullen die Bits um die Potenz nach links verschoben. Für die Rechnung $9 \cdot 4$ ergibt sich

$$1001 \cdot 2^2 = 100100$$

Mit Hilfe des Distributivgesetzes kann so die allgemeine Multiplikation von Binärzahlen auf die Addition zurückgeführt werden:

$$1001 \cdot 0101 = 1001 \cdot (2^2 + 2^0) = 100100 + 001001$$

Divisionen durch Potenzen der Zahl 2 können bei Vernachlässigung des Rests durch eine Verschiebung der Bits nach rechts realisiert werden, allerdings lassen sich damit nicht allgemeinere Divisoren realisieren. Die allgemeine Division wird auf eine Folge von Subtraktionen zurückgeführt, indem man zum Beispiel schriftlich wie im Dezimalsystem subtrahiert. Das Vorgehen ist nachfolgend am Beispiel $17 : 5 = 3$ Rest 2 gezeigt:

$$\begin{array}{rl} & 10001 \;:\; 101 \;=\; 011 \text{ Rest } 010 \\ - & \;\;0 \\ \hline & 1000 \\ - & \;101 \\ \hline & \;\;111 \\ - & \;101 \\ \hline & \;\;010 \end{array}$$

Wir sehen also, dass sich sämtliche Rechnungen durch einfache Verschiebungen und Binäradditionen realisieren lassen.

## 4. Addition mit logischen Operationen

Wir identifizieren Belegungen von Bits mit den logischen Werten falsch und wahr und versuchen, die Addition zweier Bits mit Übertrag durch einen logischen Ausdruck darzustellen. Die Wertetabelle für die Addition mit Übertrag $A + B = S$, $C$ ist in Tabelle 1 gezeigt. Die Operation $(A, B) \mapsto (S, C)$ wird als *Halbaddierer* bezeichnet.

| $A$ | $B$ | $S$ | $C$ |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |

TABELLE 1. Die als Halbaddition bezeichnete Addition zweier Bits $A$ und $B$ ergibt die Summe $S$ und den Übertrag (*Carry Bit*) $C$.

Als logische Ausdrücke verknüpfen wir die Zeilen, die den Wert 1 ergeben, jeweils mit der ODER-Operation und erhalten

$$S = (\neg A \wedge B) \vee (A \wedge \neg B) = A \vee_! B,$$
$$C = A \wedge B,$$

wobei der Ausdruck für $S$ dem ausschließenden ODER, das hier durch $\vee_!$ symbolisiert wird, entspricht.
Als nächstes betrachten wir die Addition von drei Bits $A$, $B$ und $C_{in}$, wobei $C_{in}$ ein Übertragsbit aus der Addition zweier niederer Bits sei. Dies ergibt ein Summationsbit $S$ und ein Übertragsbit $C_{out}$, deren Werte in Tabelle 2 aufgeführt sind. Die Operation $(A, B, C_{in}) \mapsto (S, C_{out})$ wird als *Volladdierer* bezeichnet.

| $A$ | $B$ | $C_{in}$ | $S$ | $C_{out}$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

TABELLE 2. Die als Volladdierer bezeichnete Addition dreier Bits $A$, $B$ und $C_{in}$ ergibt die Summe $S$ und den Übertrag $C_{out}$.

Die Wertetabelle liefert die Ausdrücke

$$\begin{aligned} S &= (\neg A \wedge \neg B \wedge C_{in}) \vee (\neg A \wedge B \wedge \neg C_{in}) \\ &\quad \vee (A \wedge \neg B \wedge \neg C_{in}) \vee (A \wedge B \wedge C_{in}) \\ &= (A \vee_! B) \vee_! C_{in} \\ C_{out} &= (\neg A \wedge B \wedge C_{in}) \vee (A \wedge \neg B \wedge C_{in}) \\ &\quad \vee (A \wedge B \wedge \neg C_{in}) \vee (A \wedge B \wedge C_{in}) \\ &= ((A \vee_! B) \wedge C_{in}) \vee (A \wedge B). \end{aligned}$$

Die Volladdition können wir mit Hilfe zweier Halbaddierer und einer ODER-Operation darstellen, wie in Abbildung 1 gezeigt ist.

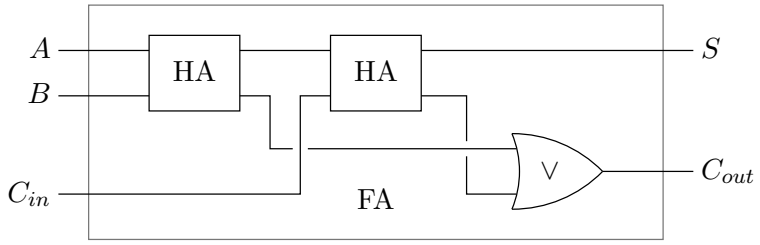


ABBILDUNG 1. Schematische Konstruktion eines Volladdierers (FA) mittels zweier Halbaddierer (HA) und einer ODER-Operation.

Sollen jetzt die Binärzahlen $a = a_k \dots a_1 a_0$ und $b = b_k \dots b_1 b_0$ addiert werden, so kann dies mit einem Halbaddierer für die niedersten Bits sowie $k-1$ Volladdierern für die Additions der höheren Bits unter Berücksichtigung der vorigen Überträge geschehen. Die Realisierung ist in Abbildung 2 gezeigt.

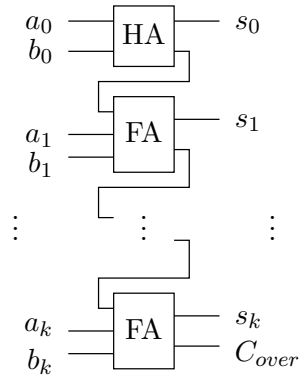


ABBILDUNG 2. Addition zweier $k$-stelliger Binärzahlen mittels eines Halbaddierers (HA) und $(k-1)$ Volladdierern (FA) unter Berücksichtigung eines *Overflow*-Bits $C_{over}$.

Die in Abbildung 2 gezeigte Schaltung ist einfach zu realisieren, aber nicht besonders effizient, da die Volladdierer jeweils das Übertragsbit des vorigen Voll- beziehungsweise Halbaddierers benötigen. Um die Rechenzeit zu verkürzen, halbiert man die Bitfolgen und berechnet die Summe der niederen Bits

$$a_{k/2} \dots a_0 + b_{k/2} \dots b_0 = [S_{low}, C_{low}]$$

sowie parallel dazu die zwei Summen der höheren Bits, einmal mit und einmal ohne Übertragsbit, das heißt

$$0 + a_k \dots a_{k/2+1} + b_k \dots b_{k/2+1} = [S^0_{high}, C^0_{over}]$$
$$1 + a_k \dots a_{k/2+1} + b_k \dots b_{k/2+1} = [S^1_{high}, C^1_{over}].$$

Die korrekte Summe ist $S^0_{high} S_{low}$ mit Überlaufbit $C^0_{over}$ falls $C_{low} = 0$ und $S^1_{high} S_{low}$ mit Überlaufbit $C^1_{over}$ andernfalls. Diese Entscheidung wird von einem sogenannten *Multiplexer* durchgeführt. Durch wiederholte Anwendung dieser Argumentation lässt sich die Rechenzeit mehrfach nahezu halbieren.

## 5. Realisierung mit Transistorschaltungen

Einzelne Bits beziehungsweise die logischen Werte falsch und wahr lassen sich sinnvoll mit elektrischen Signalen darstellen, das heißt an einer Leitung liegt entweder eine Spannung an und es fließt Strom oder nicht. Da sich jede logische Operation mittels der Grundfunktionen UND, ODER sowie NEGATION darstellen lässt, diskutieren wir nur deren technischen Realisierungen. Mit Transistoren kann über einen kleinen Stromfluss ein größerer kontrolliert werden, ähnlich wie es in Abbildung 3 für zwei Wasserkanäle illustriert ist. Liegt an der Basis eine Spannung an, so wird der Stromfluss vom Kollektor zum Emitter ermöglicht.

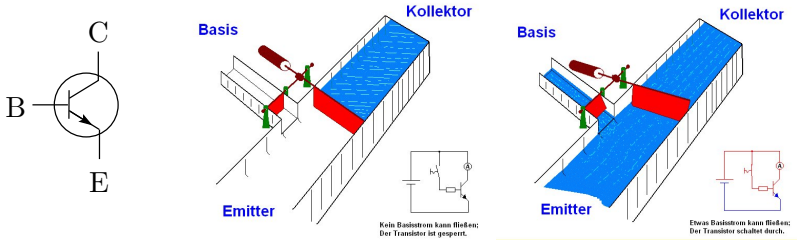


Abbildung 3. Schaltsymbol und Funktionsweise eines Transistors: ein kleiner Stromfluss von der Basis zum Emitter öffnet und schließt einen größeren vom Kollektor zum Emitter. (Quelle: Stefan Riepl (Quark48 21:02, 2. Dez. 2007 (CET)) (`https://commons.wikimedia.org/wiki/File:Transistor_animation.gif`), Transistor animation, `https://creativecommons.org/licenses/by-sa/2.0/de/legalcode`)

Abbildung 4 zeigt Transistorschaltungen für logische Grundoperationen. Die UND-Schaltung ist leicht verständlich: nur wenn an den Basen der beiden Transistoren Spannungen anliegen, das heißt nur wenn $A$ und $B$ wahr sind, kann Strom von der Spannungsquelle zur Erdung fließen und am Ausgang $A \wedge B$ liegt eine Spannung an. Bei der ODER-Schaltung überprüft man, dass

am Ausgang Spannung anliegt, sofern mindestens einer der Transistoren geschlossen ist, also Eingang $A$ oder Eingang $B$ Spannung führt. Etwas anders ist die Funktionsweise der NEGATION-Schaltung. Liegt am Eingang $A$ eine Spannung an, so fließt der gesamte Strom aufgrund des Widerstands R2 zur Erdung und es liegt keine Spannung am Ausgang an. Tatsächliche Realisierungen sind in der Regel etwas komplexer, da beispielsweise wiederholte Spannungsabfälle vermieden werden müssen.

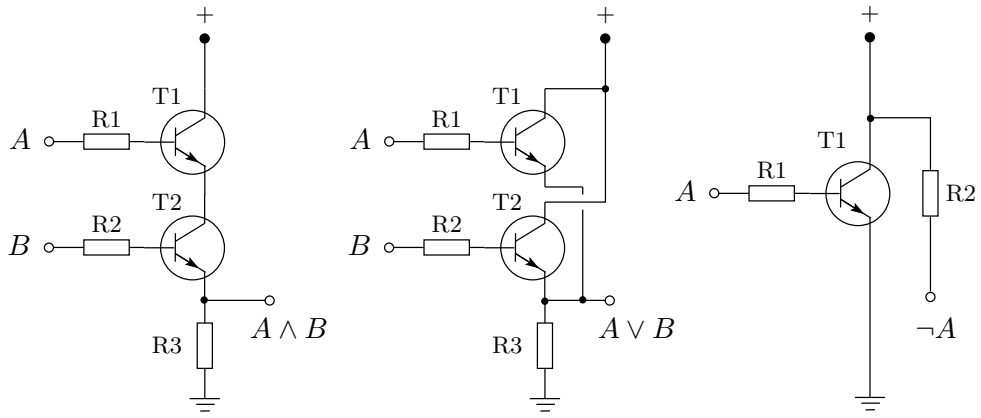


Abbildung 4. Transistorschaltungen für die konzeptionelle Realisierung der logischen Operationen UND, ODER und NEGATION.